

Software Process Improvement: Operations Perspectives

Woonghee Tim Huh*
Cornell University

November 30, 2000

Abstract

Software development is increasingly becoming important in today's businesses as an enabler of their business processes and services. However, software improvement processes are still unsatisfactory and not mature. This paper examines software process improvements from the perspective of new product development and operations management, and suggests how research in these fields may be used to guide improvement initiatives in software development.

1 Introduction

Software has become one of the most integral parts of today's corporations, and the efficiency and maintenance of business processes and organizations depend largely on the improvement and capability of their supporting software and related solutions. Despite their importance and recent advances in software engineering, software development processes have not yet been clearly understood. It is reported that one-third of software projects are canceled before completion, and more than half of the projects cost at least twice their original estimates.[1] Building on Nambisan and Wilemon's work [13] on the cross-domain knowledge sharing between new product development and software development, this paper makes a further attempt to comprehend the software development process through examination of process improvements in related fields outside the software industry, such as new product development and operations management. Examining software process improvements from the perspective of new product development and operations management, we gain insights into future direction of software development process improvements.

2 New Product Development: Human Factors

New product development (NPD) shares many similarities with software development, especially of commercial off-the-shelf (COTS) software. Both development processes go through the similar cycles of idea generation, analysis, development and commercial launch. Yet, these two fields have been isolated, and little cross-pollination of knowledge or sharing of experience has taken place. For example, a survey paper by Milson et al. [12] summarizes approaches to accelerate NPD in five categories: simplify, eliminate steps, parallel processing, eliminate delays, and speed-up. These approaches, along with documented potential benefits and limitations, may have not been explicitly spelled out in software engineering literature, but nonetheless they are very likely applicable. In a recent article, Nambisan and Wilemon [13] establish that NPD and software development can learn from each other to address similar development challenges, as they point out the difference between the emphasis of NPD literature and that of software development.

* School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853, USA. Email: huh@orie.cornell.edu. Author supported by Natural Sciences and Engineering Research Council of Canada and Semiconductor Research Council.

They name, in particular, teamwork management and accelerated product development as two examples of what software development may benefit from NPD. Whereas software development has placed emphasis on tools and technology, NPD has investigated the behavioral aspect of the interaction between business processes and people. Indeed, informal interviews with engineers from the IT department of a large corporation have shown the author that these engineers perceive that organizational factors such as communication, not technological tools, are the bottleneck in software development. Human factors such as team building, coordination and customer involvement have been absent in software development literature. Karlsson [24], for example, acknowledges the lack of software engineering studies on the “non-technical” factors such as organizational and cultural factors, but no significant changes in research or practice have been made yet.

Why then has the software development community failed to pay attention to these human factors? One possible explanation is professional training. This community is filled with engineers and mathematicians, who are used to obtaining good results through modifications in experimental procedures or clear-cut logical deductions. Although their empirical methods of physical sciences have contributed the software development process by devising new development methodologies, inventing revision-control tools, and developing the objected-oriented paradigm, they have not paid much attention to humans and their interactions, possibly due to the weak training and awareness in social sciences. This may explain why sophisticated technological solutions for software development often fail to fulfill what they promise. In contrast, the NPD community has not sought technological advances as a major source of development improvement, but instead their expertise in marketing and management has produced insights into human behavior, many of which are applicable in software development. In Figure 1, the research thrusts of the NPD community rest in the *High Management of People–Low Management of Technology* square whereas the software development community has been moving horizontally from *Ad-hoc* towards *Low Management of People–High Management of Technology*. The ultimate goal, of course, is to achieve a high performance team by excelling in both management of people as well as management of technology.

Hiring decisions and team formation for software projects have not been carefully studied in the software engineering literature. Little can be found with regards to the role of individuals and their attributes in managing and participating successful development projects, although the human capital is the biggest asset in the software industry. Software managers, in practice, make hiring decisions based on the technical toolkit of applicants, such as familiarity with software languages and hardware platforms. The NPD literature, in contrast, shows that individual characteristics are important. For example, Stevens et al. [18] show that one of the key success factors at the personnel level is creativity. They argue that creativity cannot be imposed through training, whereas business disciplines, though important, may be learned.

Communication is a challenge that software project managers need to address. A corporation’s static organizational structure does not always accommodate the dynamic nature of project teams. Often cross-functional teams are unavoidable or even desired. The word “stovepipe” is frequently used to describe inefficient intra-organizational communication. To prevent organizational barriers, some software development teams begin to use a matrix organization, consisting of traditional top-down vertical structure as well as horizontal virtual structure. Studies in NPD suggest the centrality of flexible organizations in development project success. Cooper and Kleinschmidt [4] find that a key factor in speeding up the speed to market is “an accountable, dedicated cross-functional team with a strong leader and management sponsorship.” Kessler and Chakrabarti [7] also emphasize the importance of the role of managers in setting clear schedules and time lines, as well as assigning team members who have been in the organization longer. Clarity in communication is important, and spending time in the organization helps members expand their communication channels and contacts. Their research also reports an interesting finding that the reliance on computer-aided-design systems reduces the speed, suggesting poorly-managed technology may curb the development process.

The NPD literature addresses many organizational or human aspects of development that have not been addressed in the software development literature. A close examination of such literature will generate further useful insights for improvements in software development process that current software engineering research has not identified.

3 Make-to-Order Production of Software

In the previous section, we regard software development as *development*, but we now compare it to *production*. The custom-ordered, non-COTS software development projects, are very common in corporate information technology projects. Such software projects may be viewed as an extension of make-to-order product manufacturing.

In many make-to-order environments, the reduction of cost and lead-time is achieved through a strategy called *delayed differentiation*. [8] Products are designed such that the customization of each order is performed in the late stage of the production cycle. For example, Benetton improved its operational efficiency by reversing the “dyeing” and “knitting” processes to simplify operations and to reduce lead-time of end products with multiple colors. [9] Often delayed differentiation is achieved by designing products such that they can be easily constructed by putting together pre-manufactured functional components. It is important to design interfaces such that dependencies among components are minimized and clearly specified.

Lee and Tang [8] propose three approaches for delayed differentiation: *standardization*, *modular design* and *process restructuring*. (See Figure 2.) Standardization refers to designing a component that is common to all products. In the software context, standardization may refer to producing codes that can be used to handle a variety of similar functions. For example, Gartner Group [2] estimates that the development of point-to-point application interfaces constitutes about 30 percent of the implementation costs of a major application package. If you develop inter-application communication adaptors which function with a variety of hardware platforms and file transfer protocols, then there is no need to reinvent parallel codes for combinations of platforms and protocols. This is the underlying philosophy of *middleware* solutions, which provides “plug-and-play” interfaces for many popular information systems. A similar concept called *product platform*, referring to a set of subsystems and interfaces from which a variety of products can be efficiently developed, has also been proposed for use in software products. [10]

Modular design is similar to standardization, and it means dividing a part of product into two modules where the first module is the common part, and the second module is assembled after the order becomes visible. Then, software development mainly consists of putting together a combination of modules according to functionalities. Modularization in manufacturing systems reduces cost through facilitated product design and economies of scale. Since the cost of replicating software modules is negligible, the potential benefit of modularization in software far exceeds manufacturing systems. Modularization, along with standardization, promotes software reuse, which is in fact a well-known initiative in software engineering. By reusing existing code, it is possible to reduce development effort and time as well as the likelihood of introducing bugs.

Though software reuse is an appealing conceptual idea, its effectiveness and success vary largely among projects and teams. Why can’t software development be as simple as assembly plants at Dell? If complex systems such as automobile manufacturing plants can use modularization, why can’t software development use the same components for multiple products? Software engineers point out that the inherent difficulty lies in complexity. Software construction today is compared to “having a bath full of Tinkertoy, Lego, Erector set, Lincoln logs, Block City, and six other incompatible kits - picking out parts that fit specific functions and expecting them to fit together.” ([16], [21]) Software development is perceived to be far more complex than manufacturing commodities. However, the simplification of manufacturing process has been achieved through process improvement initiatives over many years. The automobile industry, for example, has taken decades to decompose a car into sensible logical units and to make components compatible with others. In current software development, much time has been on the translation of one architectural model and syntax to another. [16] The immaturity of software development standardization and modularization is possibly due to its newness and management inexperience, [6] not necessarily due to its inherent nature. According to Baxter [24], such standardization may possibly be dictated by large organizations such as Microsoft.

Other areas of improvement for software reuse is the management of software repository or library, where reusable components are stored, maintained and retrieved. In many companies, the scope of code reuse is limited to an individual developer, and the reuse is done only by the same developer who initially created his component, consequently limiting the potential benefit of code reuse. The manufacturing analogue of software repository is inventory management. Inventory theory has been long recognized as a vital part of

procurement and distribution since 1950's (e.g. [17]), whereas many software companies surveyed by Morisio et al. [11] did not even have a repository. It is ironic that while some companies like Toys "R" Us maintain its competitive advantage from its powerful information and distribution systems for toys,[15] many software organizations become less efficient due to the lack of information on their reusable codes.

Modularization allows manufacturing companies to outsource a set of components to outsider vendors, and thus developers benefit not only from core competency of suppliers but also from risk-management (by uploading some penalty of failure to suppliers). The extent of outsourcing in software industry is currently limited to buying COTS software or hiring consultants for a comprehensive solution. With increasing maturity in modularization in software, outsourcing may be strategically employed.

Inventory systems use forecasting to determine target inventory levels. In software reuse, we would like to forecast a given functionality's reuse likelihood and occurrences in future. Such information can be used to support decisions regarding how generic software components should be made. Some research has been done on the evolution of technology in general (e.g. [20] and [22]), but they are not at the level of tailoring results for the purpose of decision making regarding software reuse. If components are generic, the initial investment cost is high, but it is more likely to be employed by future developments. This tradeoff has not yet been examined in software engineering literature, and there exists no quantitative or operational research model. Code reuse in software development still remains immature and unsteady, but with experience the industry may reap benefits of standardization and modularization, which manufacturing systems have reaped.

Process restructuring is resequencing process steps in making a product. This is an area where software engineering has indeed paid much attention, and has consequently produced different methodologies (e.g. waterfall model, iterative model, and rapid prototyping) and project management tools (e.g. CASE tool). The new challenge is the proper employment of such tools as discussed in Section 2, and incorporate them with standardization and modularization initiatives.

4 Quality Management

The importance of quality has been long recognized in manufacturing systems. Higher quality implies lower failure rate, increased productivity, and higher customer satisfaction. It has been recognized that quality is not achieved through acceptance/rejection testing, but through intentionally installing it in the design and processes. Disciplined processes result in standardized products, and this philosophy became widespread, and have been implemented by Statistical Process Control in 1980's and and through ISO 9000 certification in 1990's.

Quality control, in software industry, has traditionally been synonymous with end-of-line testing, but now new attempts have been made to introduce statistical control in software engineering. For example, Frorac and Carleton [5] promote the usage of control charts to measure and to improve performance of software processes. The Software Engineering Institute, affiliated with Carnegie Mellon University, has developed the Capacity Maturity Model (CMM) to identify characteristics of mature software development organizations.[14] This five-level model is used to assess the strengths and weaknesses of an organization's software development processes. The tenets of CMM are gaining popularity among practitioners and business writers (e.g. [23]). Many software firms, especially emerging companies in India, use the CMM certification as a part of their strategic positioning.[3] Quality and procurement are increasingly becoming important in developing competitive advantages in the software market.

In light of heavy competition for human resources, retention and acquisition of software developers have become increasingly difficult. It may prove useful to apply a widely-accepted value of customer satisfaction to employee satisfaction. In her recent book [19], Seybold stresses the importance of customer-orientation of today's electronic business, as depicted in her title *Customers.com*. Among her suggestions are: Make it easy for customers to do business with you; Focus on the end customer for your products and services; Redesign your customer-facing business processes from the end customer's point of view; and Foster customer loyalty. Recognizing that software developers are human beings and key development resources are subject to quality improvement, it is also important to create a developer-friendly business organization, which we coin

“develops.com”. (Just replace *customer* with *developer* in Seybold’s recommendations.) In high maturity organizations under the CMM, success in software development does not depend on ad-hoc triumph of heros, but rather on the systematic management of projects based on accumulated organizational knowledge. Thus the CMM-based improvements will relieve the stress of developers by reducing the occurrence of “fire-fighting” and of meeting unrealistically-scheduled deadlines, helping companies to retain their highly-valuable skill sets.

The operations management discipline teaches us that a few fundamentals need to be established for successful CMM-based or other quality initiatives. Collecting data is important to baseline current practices to be compared with future improvement initiatives. Yet many software development teams are uncertain as to what metrics are relevant and thus need to be collected. We need to define, quantitatively, what *quality* and customer acceptance mean, and what levels need to be attained. We also need to quantify productivity and resource management. For example, one manager told the author that the capability of his team is determined simply by the number of developers, with total disregard to individual experience or competence. Similarly, the number of lines of code produced is not a good indicator of productivity. Quantification of quality and other metric definitions are both necessary for quality improvement and important because the reward structure drives the behavior and culture of any organization. (“What is measured gets done.”)

Though the ideals and promises of CMM are sound, they are not a magic answer to all problems in software development. Total Quality Management, a process-oriented improvement initiatives based quantitative measurement, has produced a mix of successes and failures, in factory systems. Most of the failures may be attributed to the lack of organizational support, inefficient implementation and improper metrics, implying blind implementation of the CMM or other quality initiatives may similarly result in unfruitful waste of time and money.

5 Conclusion

We compare software process improvements to other process improvements in new product development and operations management. We consider such areas as organizational factors, code reuse and quality management. Software engineering has produced process improvement initiatives similar to those in operations-related fields, and thus may benefit further from their research and practice.

Acknowledgment

The author would like to thank Russ Hargrave and his staff at Intel Corporation as well as Professor Robin Roundy at Cornell University for motivating and supporting this research.

References

- [1] Chaos. Technical report, The Standish Group, West Yarmouth, MA, 1995.
- [2] An ROI model for enterprise application integration. Technical report, Gartner Consulting, San Jose, CA, april 1999.
- [3] Erran Carmal. *Global Software Teams: Collaborating Across Borders and Time Zones*. Prentice Hall, 1999.
- [4] Robert G. Cooper and Elko J. Kleinschmidt. Determinants of timeliness in product development. *Journal of Product Innovation Management*, 11:381–396, 1994.
- [5] William A. Florac and Anita D. Carleton. *Measuring the Software Processes: Statistical Control for Software Process Improvement*. Addison Wesley, 1999.
- [6] William B. Frakes and Christopher J. Fox. Quality improvement using a software reuse failure modes model. *IEEE Transactions on Software Engineering*, 22(4):274–278, April 1996.

- [7] Eric H. Kessler and Alok K. Chakrabarti. Speeding up the pace of new product development. *Journal of Product Innovation Management*, 16:231–247, 1999.
- [8] Hau L. Lee and Christopher S. Tang. Modelling the costs and benefits of delayed product differentiation. *Management Science*, 43(1):40–53, January 1997.
- [9] Hau L. Lee and Christopher S. Tang. Variability reduction through operations reversal. *Management Science*, 44(2):162–172, February 1998.
- [10] Marc H Meyer and Robert Seliger. Product platforms in software development. *Sloan Management Review*, 40(1), 1998.
- [11] Maurizio Morisio, Michel Ezran, and Colin Tully. Introducing reuse in companies: A survey of european experience. In *Proceedings of the Fifth Symposium on Software Resusability*, pages 3–9, Los Angeles, CA, USA, May 1999.
- [12] S. P. Raj Murray R. Millson and David Wilemon. A survey of major approaches for accelerating new product development. *Journal of Product Innovation Management*, 9:53–69, 1992.
- [13] Satish Nambisan and David Wilemon. Software development and new product development: Potentials for cross-domain knowledge sharing. *IEEE Transactions on Engineering Management*, 47(2):211–220, May 2000.
- [14] Mark C. Paulk, Charles V. Weber, Bill Curtis, and Mary Beth Chrissis. *The Capacity Maturity Model: Guidelines for Improving the Software Process*. Addison Wesley, 1995.
- [15] James Brian Quinn and Frederick G. Hilmer. Strategic outsourcing. *Sloan Management Review*, 35(4):43–55, 1994.
- [16] Alexander Ran. Software isn’t built from lego blocks. In *Proceedings of the Fifth Symposium on Software Resusability*, pages 164–169, Los Angeles, CA, USA, May 1999.
- [17] Herbert Scarf. The optimality of (S, s) policies in the dynamic inventory problem. *Mathematical Methods in the Social Sciences*, pages 196–202, 1959.
- [18] Greg Sevens, James Burley, and Richard Divine. Creativity + business discipline = higher profits faster from new product development. *Journal of Product Innovation Management*, 16:455–468, 1999.
- [19] Patricia B. Seybold. *customers.com: How to Create a Profitable Business Strategy for the Internet and Beyond*. Time Books,, 1998.
- [20] Chitra Sharma, A. D. Gupta, and Sushil. Flexibility in technology forecasting, planning, and implementation: A two phase idea management study. In *Precedings Vol-2: Papers Presented at PICMENT ’99*, Portland, OR, USA, July 1999.
- [21] Mary Shaw. It’s not just functionality, it’s the packing. In *Proceedings of the Symposium on Software Resusability*, Seattle, WA, USA, April 1995.
- [22] Clement Wang, Xuanrui Liu, and Daoling Xu. Chaos theory in technology forecasting. In *Precedings Vol-2: Papers Presented at PICMENT ’99*, Portland, OR, USA, July 1999.
- [23] Sami Zahran. *Software Process Improvement: Practical Guide for Business Success*. Addison Wesley, 1998.
- [24] Mansour Zand, Vic Basili, Ira Baxter, Martin Griss, Even-Andre Karlsson, and Dewayne Perry. Reuse R&D: Gap between theory and practice. In *Proceedings of the Fifth Symposium on Software Resusability*, pages 172–177, Los Angeles, CA, USA, May 1999.

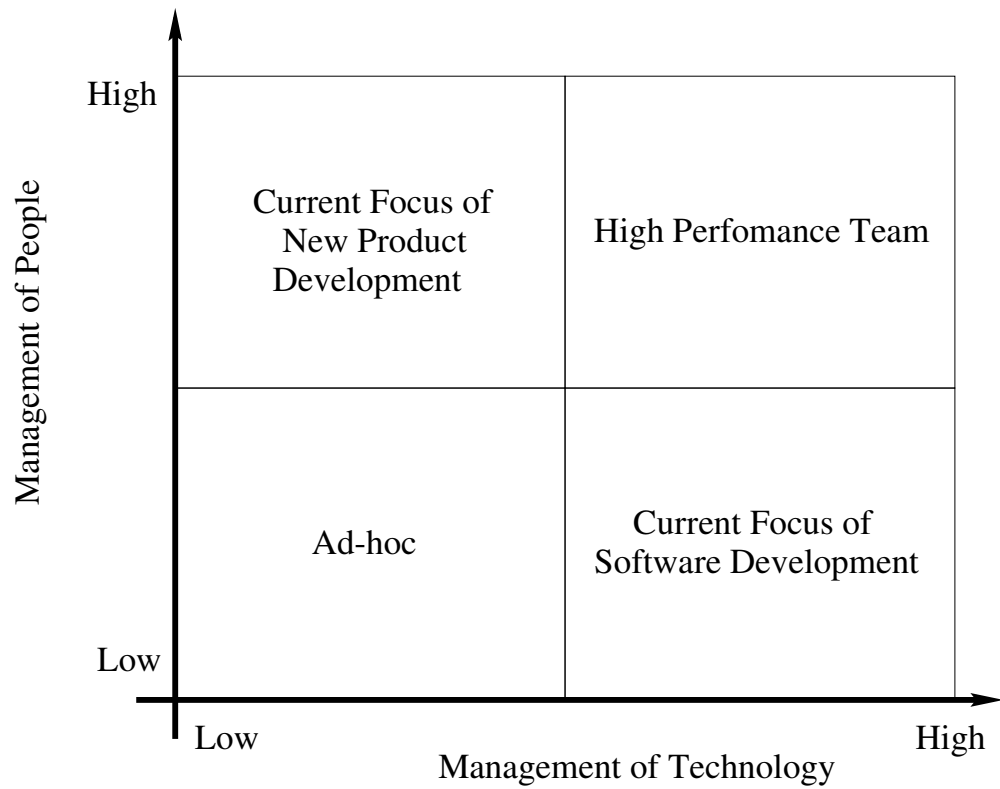


Figure 1: Matrix of Management of People vs. Management of Technology

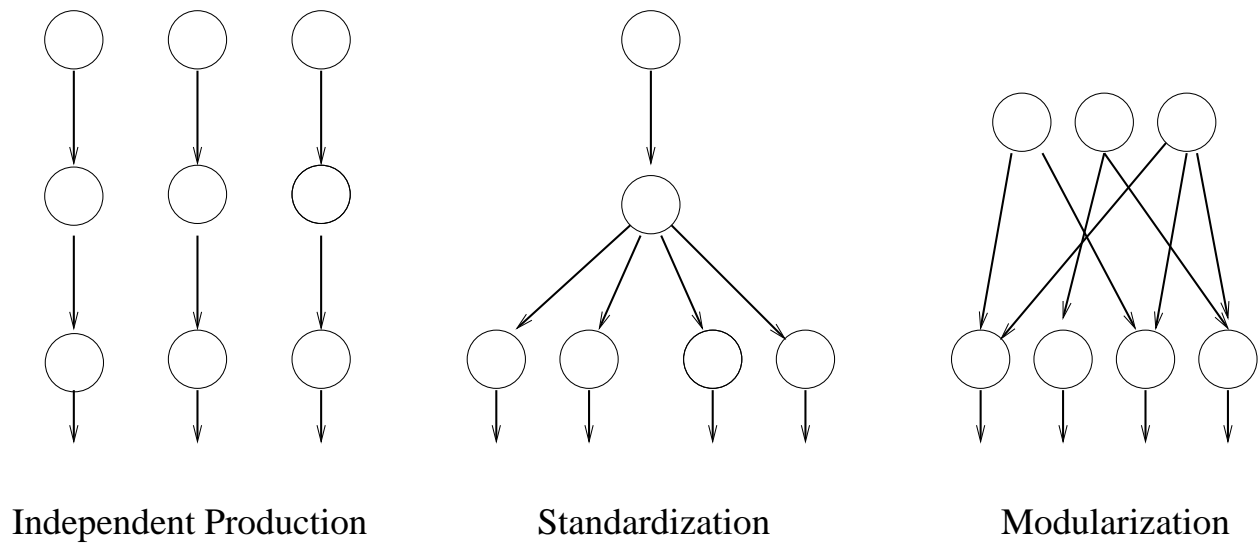


Figure 2: Schematics of Standardization and Modularization